SCHOOL OF
COMPUTING

Title: The history of programming language semantics: an overview

Names: Troy K Astarte

TECHNICAL REPORT SERIES

No. CS-TR- 1533 June 2020

TECHNICAL REPORT SERIES

No. CS-TR- 1533 Date: June 2020

Title: The history of programming language semantics: an overview

Authors: Troy K. Astarte

Abstract: In the 1960s, a full formal description was seen as a crucial and unavoidable part of creating a new programming language. A key part of that was a thorough and rigorous description of the semantics. However, in the decades since, the focus on providing this has somewhat diminished. Why was formal semantics once seen as so critical? Why did it not succeed in the ways hoped? These questions are explored by considering the history of model-based approaches to describing programming languages, with a particular focus on the IBM Laboratory Vienna under Heinz Zemanek, and the Programming Research Group at Oxford University under Christopher Strachey. It is shown that there were a variety of different approaches to the problem, stemming from the different backgrounds of the people and groups involved. The story of formal language description is in some ways also the story of early programming languages and the integration of mathematics into the emerging new field of computer science, resulting in the formation of theoretical computing in the European style. This paper is the first draft of one that will be submitted for publication in a Johns Hopkins University Press volume. The finished paper is likely to be significantly different to the one here. This note will be updated when bibliographic information is available for the published version.

June 2020 Newcastle University.
Printed and published by Newcastle University,
School of Computing, Urban Sciences Building,
Newcastle upon Tyne, NE4 5TG, England.

Bibliographical Details: The history of programming language semantics: an overview

Title and Authors : Troy K. Astarte

NEWCASTLE UNIVERSITY
School of Computing. Technical Report Series. CS-TR- 1533

Abstract: In the 1960s, a full formal description was seen as a crucial and unavoidable part of creating a new programming language. A key part of that was a thorough and rigorous description of the semantics. However, in the decades since, the focus on providing this has somewhat diminished. Why was formal semantics once seen as so critical? Why did it not succeed in the ways hoped? These questions are explored by considering the history of model-based approaches to describing programming languages, with a particular focus on the IBM Laboratory Vienna under Heinz Zemanek, and the Programming Research Group at Oxford University under Christopher Strachey. It is shown that there were a variety of different approaches to the problem, stemming from the different backgrounds of the people and groups involved. The story of formal language description is in some ways also the story of early programming languages and the integration of mathematics into the emerging new field of computer science, resulting in the formation of theoretical computing in the European style. This paper is the first draft of one that will be submitted for publication in a Johns Hopkins University Press volume. The finished paper is likely to be significantly different to the one here. This note will be updated when bibliographic information is available for the published version.

About the authors: Troy Kaighin Astarte is a research assistant in computing science at Newcastle University. Troy's PhD research was on the history of programming language semantics. They are now working on a research project on the history of concurrency in programming and computing systems, funded by the Leverhulme Trust.

Suggested keywords: Keywords: history of computing, history of computer science, programming languages, formal semantics.

The history of programming language semantics: an overview

Troy K. Astarte*
Newcastle University
troy.astarte@ncl.ac.uk

June 29, 2020

Abstract

In the 1960s, a full formal description was seen as a crucial and unavoidable part of creating a new programming language. A key part of that was a thorough and rigorous description of the semantics. However, in the decades since, the focus on providing this has somewhat dimin- ished. Why was formal semantics once seen as so critical? Why did it not succeed in the ways hoped? These questions are explored by consid- ering the history of model-based approaches to describing programming languages, with a particular focus on the IBM Laboratory Vienna under Heinz Zemanek, and the Programming Research Group at Oxford Uni- versity under Christopher Strachey. It is shown that there were a variety of different approaches to the problem, stemming from the different back- grounds of the people and groups involved. The story of formal language description is in some ways also the story of early programming languages and the integration of mathematics into the emerging new field of com- puter science, resulting in the formation of theoretical computing in the European style.

This paper is the first draft of one that will be submitted for publication in a Johns Hopkins University Press volume. The finished paper is likely to be significantly different to the one here. This note will be updated when bibliographic information is available for the published version.

1 Introduction

In Amsterdam, in October 1970, the International Federation for Information Processing (IFIP) celebrated its tenth anniversary. The introductory address was given by Friedrich Bauer, who sketched the history of computation from ancient methods of counting using pebbles and coins right

*This work was supported in part by an EPSRC doctoral studentship and in part by a Leverhulme Trust grant.

up to the publication of ALGOL 68 [Bau72]. This fully formalised specification, which covered the language's syntax, abstract interpretation, and context dependencies, was depicted by Bauer as a pinnacle of achievement in computing: how far the field had come since its humble beginnings! [1] Indeed, as early as 1964, the group of language designers working on ALGOL 60's successor agreed unanimously that the new language should be "defined formally and as strictly as possible in a meta-language"[Utm64a, pp. 19–20]. (They were considerably less unanimous about the *choice* of meta-language.)

Now, however, it is a very rare language that contains a complete specification of syntax, semantics, and context-dependencies. Mosses lists some examples [Mos01, pp. 183–4] but these are quite few and far between. Why did such an air of optimism surround language description in the 1960s and 1970s—enough to convince the skeptical Bauer? Why and how did this optimism die out? How did the researchers and practitioners of formal language description view their work: what were their motivations and goals, and how did these become realised? This—specifically, on the semantics part of language description—was the topic of the author's PhD research [Ast19], and in the current paper, major findings of this work will be reported.

While technical factors obviously had a major factor on the ways various approaches to semantics were developed, there was at least as much impact from other directions, particularly the intellectual backgrounds of the historical actors involved. Business and research pressures from institutions also affected the work, as well interpersonal and interorganisational interactions and dynamics. This paper will focus on these, rather than the technical aspects. An exploration of the technical challenges faced by semanticists can be found in another paper from the same au- thor [JA18]; a further companion piece looks specifically at formal models of one particular programming language, ALGOL 60, both contextually and technically [AJ18].

The current paper begins with Section 2 which contains a brief overview of the history of programming, especially as it relates to concerns that were relevant for formal semantics work. Next, there is a short explo- ration of the nature of formalism, and various reasons for its application. This leads on to a discussion of what was meant by formal descriptions of programming languages. Section 3 looks at the challenges of formal definition and explores some of the solutions. In Section 4, the motivations of the various historical actors in question is considered. Section 5 examines the backgrounds of the people involved and how those affected their work. Section 6 argues for the importance of collaboration in the works discussed. In Section 7 the various criticisms levelled at formal semantics are explained. Section 8 makes the case for formal semantics as a crucial part of the development of computing theory in Europe. Section 9 looks at the impact of formal semantics work on other areas of computing. Finally, Section 10 concludes the paper.

[1] Bauer's own view on formal language specification had developed too: in 1963, at a meeting of IFIP's Technical Committee 2, he had argued against the creation of an IFIP sponsored WG on the subject [Utm63, p. 7].

The main focus of the research summarised here is on two centres of work: the Oxford University's Programming Research Group (PRG), and IBM's Vienna Laboratory (VAB). The PRG was run by Christopher Strachey, who led the denotational semantics effort alongside the logician Dana Scott, assisted by Joe Stoy. The Group was also noteworthy for having significant contributions from graduate students, many who became members of staff: Chris Wadsworth, Peter Mosses, and Robert Milne. The VAB worked under the direction of Heinz Zemanek, with Kurt Walk managing. The major intellectual contributors were Peter Lucas, Hans Bekič, Cliff Jones, Dines Bjørner, and Wolfgang Henhapl. Other stories are also addressed outside these two centres, including the work of John McCarthy, Peter Landin, Adriaan van Wijngaarden, Gordon Plotkin, and others. The time period in focus is roughly from 1960 to 1980, which is to say from around the beginning of computer language formalisation to a period in which there was a shift in direction in theoretical computer science.

2 Background

This is not the place for a full history of computing machines[2] or even programming languages,[3] but the topic at hand is clearly linked to both. It is worth providing a brief overview.

Very early computers were controlled directly with hardware, through wiring, switching, and plugging. Once the stored program paradigm took root, instructions needed to be encoded in a machine-readable format. These 'order codes' tended to be simple, with a paucity of commands, and a direct correspondence with machine operations. Peláez points out that this meant that programs were written to fit the peculiarities of their target machines [PV88, p. 4]. These early programs were not intended to be easy for humans to read, and were often encoded numerically on paper tape or in punched cards. From the early '50s more complex systems for encoding programs appeared: these came under many different names, such as as Autocode [Gle52] or Speedcoding [Bac54].

Later, 'high-level languages' appeared, the most important (for the present story) of which was ALGOL.[4] It is tied closely to the history of semantics, firstly because early versions of the language were presented with a formalised syntax (but informal semantics); secondly because ALGOL 60's presentation as machine-independent made the definition document the ultimate reference; thirdly because it was the subject of many different styles of semantic model;[5] and finally because it represented a

[2] See [CKA96] for a thorough history of the computing machine.

[3] A long-running conference series, *History of Programming Languages*, provides plenty of material; Priestley dedicates a chapter (8) of [Pri11] to the topic; Section 2.2 of the author's thesis is a lengthier treatment than the current outline.

[4] Details about the history of ALGOL can be found in the *History of Programming Languages* papers of Perlis [Per81] and Naur [Nau81]. The special issue (Vol. 36, No. 4) of *The Annals of the History of Computing* provides deeper historical insights. See also Section 2.3 of [Ast19].

[5] See [AJ18].

paradigm shift in computing towards making programming languages an important object of study [Pri11, p. 229].

Early coding systems generally had very clear connections to their target machines and were notations whose primary purpose was to save the programmer time. The 'meaning' of such languages tended to be fairly easily understood (even if the notation was somewhat baroque) as it was composed of machine functions in a bottom-up fashion. This can be seen in, for example, the method of presentation of the EDSAC order code which simply lists each order's effect on the machine's components.

Walk [Wal02, p. 80] argued this point:

> With machine and assembler languages, whose structure was fairly simple, programs consist[ed] of equally formed statements whose meaning was determined by the status and the changes of the hardware components they referred to.

Priestley [Pri11, Section 7.9] builds the same argument, showing that for Goldstine and von Neumann, meaning of programs flowed directly from the program text and its relation to the machine.

High-level languages represented a significant departure from this, although as the shift happened gradually it was not immediately obvious. The greater abstraction of FORTRAN, ALGOL, and other successor languages allowed much easier formulation of programs: meaning was constructed in a top-down approach. At this period, the late 1950s, most advanced programming was performed for scientific purposes, and encoded mathematical calculations. This made mathematical concepts the obvious basis for developing semantics. However, the new conception of 'program' as a more abstract object was disconnected from the functioning of the machine. If the program was written in terms of abstract entities and its execution was understood in terms of hardware function, could a truly believable correspondence between the two be established?[6] The important point to note is that if the description of a language could no longer be based on machine functions, it needed some other basis—which is where formalism, particularly semantics, came in.

There was another problem with high-level languages: to actually run, they had to be brought down to the level of the machine. This was often achieved with the help of another program that converted a high-level program into machine code. These are now typically called 'compilers' after Hopper's original term [Hop81], but terms such as 'interpreter routine' or 'automatic programming' (to refer to the whole process) [WWG51] were also common. Introducing a program to manage the translation brought a risk, however:

> It was bad enough to have a slow, inefficient and bug-ridden daily production of software, but [bad programmers] really came into their own when they wrote slow, inefficient and bug-ridden compilers which produced deformed object code, which confused the hapless programmer endlessly. How could he resolve whether the problem lay with his lack of skill, or the operating system, or the compiler, or even with the intermittent hardware fault?

[6] This concern was to cause some philosophical problems later; more on that in Section 7.

([Fee81, p. 265]; [quoted in PV88, p. 117])

This concern about the introduction of bugs was typical of worries about the reliability of programs, which came to a head in the late 1960s with the declaration of a 'software crisis'. Pelàez provides a comprehensive historical treatment of this area in [PV88]; MacKenzie [Mac01, Ch. 2, 8] and Tedre [Ted14, Ch. 4] write specifically about the impact of this time period on formal computing. In brief, as programs became more pervasive throughout society, getting them correct was seen as increasingly important, especially as they controlled extremely critical systems.[7] There did not, however, appear to be a commensurate increase in the skill of programmers in the workforce or success in software projects; quite the reverse, in fact. Something had to be done to address this worrying situation, and one approach, popular amongst computing academics, was that mathematical techniques should be used to state and discharge proofs about programs. A crucial part of this was the careful specification of the program, which required the language itself to be clearly understood.

The use of formalisation was critical here: precision and unambiguity were seen as requisites. This reflects a view similar to Bertrand Russell's on mathematics when he wrote "Ordinary language is totally unsuited for expressing what physics really asserts, since the words of everyday life are not sufficiently abstract. Only mathematics and mathematical logic can say as little as the physicist means to say" [Rus31]. The desire was for a description technique which was abstract and mathematical in nature. Milne and Strachey [MS74, pp. 10–1] wrote:

> The use of 'high-level' programming languages encourages programmers to think in terms of the abstract objects being manipulated rather than the operations which computers perform on bit patterns. This means that we should be able to describe a program in terms of the abstract objects it uses; this is the genesis of a need for a theory of programming language semantics.

This was echoed by Marcotty, Ledgard, and Bochmann [MLB76, p. 274]:

> It is precisely in the context-sensitive and semantic areas that formalism is needed. There is generally little argument over the precise syntax of a statement even if there is no for- mal description of it. All too often, however, an intuitive understanding of the semantics turns out to be woefully superficial. It is only when an attempt at implementation (which is, after all, a kind of formal definition) is made that ramifi- cations and discrepancies are laid bare. What was thought to have been fully understood is discovered to have been differ- ently perceived by various readers of the same description. By then, it is frequently too late to change, and incompatibilities have been cast in actual code.

Zemanek, manager of the VAB, quoted Bertrand Russell's remarks about formalism in mathematics to justify the Viennese work on pro-

[7] Both Tedre [Ted14, p. 63] and MacKenzie [Mac01, pp. 23–4] give the example of the US Military's Ballistic Missile Early Warning System malfunctioning by mistaking the rising moon for a Soviet missile attack.

gramming languages [Zem65, p. 141]:

> Russell once gave four reasons for formalization which still apply to both theory and practice: (1) security of operation is assured, (2) tacit pre-assumptions are excluded, (3) notions are clarified, and (4) resolving structures can be applied to many other problems.

There were other traditions in computing, such as engineering, which took other views on the best way to address program reliability. But the sort of thinking seen in the quotations above was typical amongst the class of computing academic who had been trained in mathematics. For them, careful specification of programming languages was a deeply important goal.

The term 'semantics' to refer to the meaning of a programming language, particularly when discussing its formalisation, came in fairly early, appearing prominently and influentially in Backus' paper on ALGOL 58 [Bac59]. One reason for this could be that the use of 'language' as a metaphor for describing the encoding of programs for computers, as discussed by Nofre, Priestley, and Alberts [NPA14], brings in concepts from linguistics. Morris [Mor46] had split natural language understanding into 'syntax', 'semantics', and 'pragmatics', and starting from the 1950s linguists such as Chomsky were applying notions of formality and mathematics to analyse human language (see, e.g., [Cho56]). As can be seen from the papers at the 1964 *Formal Language Description Languages for Computer Programming* conference [Ste66], these ideas were a central part of the way computing academics tried to get to grips with programming languages. The influence of linguistics on this area of computing was mentioned by Lucas [Luc78, p. 3], who wrote "By viewing computers as language interpreting machines it becomes quite apparent that the analysis of programming (and human) languages is bound to be a central theme of Computer Science".

Trying to make definitions about historical concepts is always problematic,[8] but it is important at least to delimit an area of interest for a historical study. So, exactly what kind of 'semantics' is being discussed here? The various historical actors involved had different beliefs of exactly what 'semantics' meant to them, which are explored throughout [Ast19]. One early definition was given by Dijkstra [Dij61]:[9]

> As the aim of a programming language is to describe processes, I regard the definition of its semantics as the design, the description of a machine that has as reaction to an arbitrary process description in this language the actual execution of this process. One could also give the semantic definition of the language by stating all the rules according to which one could execute a process, given its description in the language. Fundamentally, there is nothing against this, provided that nothing is left to my imagination as regards the way and the

[8] Mahoney [Mah96] particularly cautions against redefining concepts from the past in modern terminology.

[9] Dijkstra would later eschew such a view, preferring a property-based approach to language understanding than a mechanistic or denotational one [Dij74].

> order in which these rules are to be applied. But if nothing is left to my imagination I would rather use the metaphor of the machine that by its very structure defines the semantics of the language. In the design of a language this concept of the "defining machine" should help us to ensure the unambiguity of semantic interpretation of texts.

This idea of using a machine to define a language is at the core of 'operational' approaches to semantics. The other main approach considered in this work is the 'denotational' method, a basic definition of which is given by Plotkin [Plo18]: "it's just the ascription of suitable abstract entities to suitable syntactical entities".

Having set the background and made clear with what the current paper is concerned, let us begin to look at findings from this research into the history of formal semantics.

3 Challenges and solutions

The first obvious finding from the reported research is that the task of defining the semantics of programming languages proved to be a difficult challenge.[10] As pointed out by Strachey, a very central part of programming languages was that the values associated with variables change over time, a property he termed 'referential opacity' [Str66, p. 201]. This represented a major shift in complexity from standard mathematics, where a variable represented a quantity that, while unknown, had a fixed value. This made the task of writing the semantics of programming languages a markedly tougher one than that faced by Tarski [Tar44] or Kripke [Kri63] when they wrote semantics of mathematics. Furthermore, the large-scale programming languages of the 1960s and 70s tended to contain a great deal of complicating features, including blocks, procedures, and jumps; and, worst of all, concurrency.[11] Perhaps the best example of a language rich in features was IBM's PL/I. Dijkstra [Dij72, p. 862] described it as "like flying a plane with 7,000 buttons, switches, and handles to manipu- late in the cockpit"[12]—and the unfortunate Vienna Lab had to deal with this language its entirety.

As semantics researchers dealt with increasingly complex language features, so the complexity of the semantics they produced increased. John McCarthy, for example, started his semantics work in the early 1960s with a very simple definition of a state and functions for transforming it, but as he begun to include jumps, and particularly notions of compiler correctness, the descriptions became much larger. Proofs about these

[10] For more on the challenges to semantics inherent in programming languages, and an overview of the responses from the main schools of semantics, see [JA18].

[11] Starting from the late 1950s, computing machines were built that could execute multiple tasks in parallel, but with shared resources such as memory, interference between tasks had to be managed very carefully. This caused many problems for programmers trying to exploit the advantages of concurrency. Problems for semantics were more fundamental: a concurrent program can have many correct answers, and so a program is no longer a one-to-one correspondence between input and output. Pnueli gave a good overview of the reasons why this posed inherent challenges to standard semantic approaches [Pnu79, pp. 45–7].

[12] For more on PL/I, see [Ast19, § 5.3]

definitions—a key goal for McCarthy, as will be seen later—were commensurately large, and grew disproportionately. A twofold increase in size of compiler description between 'Micro-ALGOL' [MP67] and 'Mickey' (a slightly richer language, worked on by McCarthy's student James Painter) [Pai67] led to a twenty-fold increase in proof length. Landin had to rewrite the texts of programs and then extend their interpreting machine with another operator (alongside other tricks) to handle jumps in his style (see [Lan66b; Lan65a; Lan65b]). In the denotational approach, increasing numbers of parameters were passed to the semantic function to cope with extra features.

It was precisely to this problem that Hoare referred to during discussion of his axiomatic method at Working Group 2.2 [in Wal69]:

> Walk: Has the application of the method so far been restricted to languages for which the state of the computation can be described in terms of explicit names of the program, i.e. where no entities are manipulated that have no names at the language level?
>
> Hoare: Yes. But, of course, difficult things are difficult to describe.
>
> Strachey: What is "difficult" very much depends on the frame-work of thinking. For example, assignment is difficult in the λ-calculus approach, recursion is difficult in other systems. But both occur in programming languages and are simple to use.

As Strachey pointed out, a different approaches could find challenges easier or harder, and consequently the focus on a particular challenge affected the style of approach taken. Strachey himself was very keen on producing a *mathematical* (his term) semantics, and so wanted to use functions as the base for his definitions, as they were well-understood mathematical objects. That said, there were a number of core concepts which emerged repeatedly in different styles. Occurring most frequently was the idea that an abstract representation of the 'state' of computation was the key semantic object, as well as the notion of defining a command by how it changed the state.[13] An argument could even be made, as it was by Dan Berry in a (2019) email to the current author, that the differences between operational and denotational semantics of all styles were essentially "notational"; at the very least, conversion between such specifications could be achieved relatively easily.

Although approaches to semantics changed across the period in focus, very clear influences from some early practitioners permeated. McCarthy's concepts of state, an abstract representation of syntax, and the translation of statements into interpretation functions, continued to af- fect both operational and denotational semantics. Landin's idea of an abstract interpreting machine was central to the VAB's definition style, and his lambda calculus[14] framework likely had an important impact on

[13] Axiomatic semantics, after Hoare [Hoa69], had no *explicit* state, but predicates were defined over an implicit state of computation, and furthermore displayed the same 'before and after command' notion.

[14] The lambda calculus, proposed by Church [Chu41], comprises a notation for mathematical functions, and a series of rule for conversion of terms in the notation. Subsequent research

Strachey. Rod Burstall, too, had a number of very important ideas in the late 1960s, including (according to Scott [Sco00]) the idea, central to denotational semantics, that a store could be a function from locations to contents. Burstall also supervised Plotkin during the latter's creation of structural operational semantics (SOS).[15]

4 Exploring motivations

Different semantics workers tended to put emphasis on different language challenges, and this was often related to their motivation for attempting language definition work in the first place. The motivations of these practitioners tended to fall into two groups, broadly defined as 'theoretical' and 'practical'. (One exception is Plotkin, who developed the SOS style for primarily educational reasons). Most semanticists expressed a desire to contribute to both aspects, but generally tended to favour one over the other. Theoretical motivations were usually about formalising the concept of computation; practical ones about developing good things: languages or compilers.

McCarthy, in his early writings on semantics, expressed a desire to find a place for computation within mathematics. As Tedre [Ted14, Ch. 3] explained, while logicians and mathematicians such as Gödel and Turing had worked on foundational computation problems in the 1930s, the introduction of practical computational machines in the 1940s and 1950s created a different kind of problem: what exactly could these computers *do*? McCarthy [McC61; McC63] wrote that he wanted to find the basic notions of computation and then explore what deductions could be made from there. A key part of this was finding a formal notation to express abstract computation—in other words, a formal semantics of programs. Tedre characterised it as following: "McCarthy's work on formal semantics ultimately aimed at the ability to prove that an executable program will work exactly as specified." [Ted14, p. 155]. (Mahoney [Mah02] also reported this work of McCarthy). Strachey approached a similar prob- lem from a different angle: he wished to understand precisely what was happening when one writes and runs a program, and formal semantics provided a tool for that.

Mahoney [Mah02] wrote that the use of programming languages was central to understanding the powers and abilities of computers, and the importance of understanding these languages led to the necessity of formal tools for discussing them—in a separate paper, he explicitly mentioned denotational semantics as part of "the mathematics that makes the [computer] theoretically possible" [Mah88, p. 118]. Milner [Mil93] had similar beliefs, saying "this study of formal language for controlling this prosthetic limb[16] which is the computer is just completely fundamental".

showed it to have an expressive power equivalent to Turing machines. For a history of lambda calculus, see [CH06].

[15]Burstall was also very influential in other areas of computing for his notions of structural induction, intermittent assertions, and program transformation.

[16]The use of terms like Milner's which anthropomorphise computing and borrow notions from 'assistive' or 'accessibility' tech are interesting and perhaps problematic. There is a large

Arguments for formally specifying and verifying programs were part of the academic computing zeitgeist in the 1970s, as DeMillo and Lipton—critics of formal verification—stated in an interview: "It's difficult to imagine now, but the amount of influence that formal verification had in computing at the time was enormous. It was *the* only way to under- stand programs; it was *the* only way to understand the foundations of programming" [quoted in Ted14, p. 67. Emphasis original]. So the desire to apply these apparently rigorous tools and techniques to programming languages was a crucial part of computing—at least in *this* conception of it.

An early desire for formal semantics is that it would compensate for the growing separation between programs and physical machines; as is discussed in this paper, higher level languages did not have immediate correspondence with machines, and so an alternative ontological base could be helpful. In 1964, Strachey had been trying to develop the CPL language and a compiler alongside it, and discovered just how tricky that could be. He wrote at the end of that year that there remained "a rather vague feeling of unease" and that he and the rest of the team were "not altogether happy that [they had] really got to the bottom of the concepts involved" [Str66, p. 216]. This led him to work on formal semantics.

Linked to this was the idea that writing a formal definition would help in the design of a better programming language: as John Reynolds pointed out, areas of the language which could be improved were often highlighted by applying the tools of formal semantics [in JMSR04]. He argued that this should be done early in the process of language design, quipping "semanticists should be the obstetricians instead of the coroners of programming languages". Schmidt [Sch00, p. 89] wrote that Strachey had pushed for this, and "led a 'second wave' of language-design research, where emphasis shifted from syntactic structure to semantic structure". Despite this, the creation of a full language description prior to its implementa- tion was extremely rare. ALGOL 68 is a notable example [WMPK69], but it was widely criticised; Standard ML [HMT87] was developed from a formal definition but was based on an existing language, ML. Jones sug- gested that perhaps the reason for this was that semantics workers hadn't created a literature that was actually useable by and useful to language designers [in JMSR04]. Plotkin echoed this, explaining that engaging with this formal language description was simply too hard for most language designers [Plo18].

For McCarthy, one major use for formal semantics was in creating a specification against which a compiler could be compared for correct- ness [MP67]. Indeed, his work with Painter shows some limited success in stating a formal equivalence between a formal language description and an abstract representation of a compiler. However, even for the simple language they used, the proofs were long and involved, and this prob- lem only grew with the size of the language involved. The Vienna Lab

---

and growing literature on technology and dis/ability (see for example [Han07; FF12; Pet15]) but this is not the place for any more than a simple remark. Notice, though, that by reframing the computer as a tool for aiding human understanding, notions that originate in the study of human understanding—such as linguistics—become candidates for use.

experienced similar difficulties with their attempts to use their evolving formalisms to prove properties about compilers. Walk [Wal02, p. 82] wrote that while they had demonstrated "practical feasibility" for proofs about a compiler, they became "unsurmountably [sic] lengthy and tedious" when attempted on a full-size language like PL/I. That said, one of the Group, Dines Bjørner, did manage to lead the development of an Ada compiler based on a formal semantics [as reported in BH14].

A final and significant reason for writing formal semantics was that it created a system that enabled reasoning about a language and programs written in that language. This was a goal of McCarthy [McC63, p. 6] who wrote that he wanted to be able to bring statements about programs within the realm of mathematical proof. For Strachey, the relative ease of reasoning about functions was a large motivator for developing a heavily mathematical approach to semantics based on functions.

The theory/practice dichotomy discussed is useful in considering the different motivations, but is somewhat artificial. While McCarthy wrote about wanting to contribute to the abstract science of computation, he also developed an approach for reasoning about the correctness of compil- ers. Jones started his work on formal description by proving some proper- ties of compilers based on a formal description, but was heavily involved in the abstract theory of the semantics. In a retrospective talk about the PRG, Stoy [Sto16a] clearly demonstrated contributions from PRG mem- bers in both areas. This fits with Strachey's vision for the group: in a memorial piece, Hoare [Hoa00] writes that he found a report in Strachey's desk upon taking over as head of department. In it, Strachey had called practical work performed without knowledge of fundamental principles "unsound and clumsy" and theory work with no connection to practical programming "sterile". This was particularly typical of computer science in the 1960s and 70s: as pointed out by Tedre, many computing academics were theoreticians hoping to build a practical science of computing by es- tablishing a theoretical base and working upwards [Ted14, p. 155]

5 Personal backgrounds

One major contributory factor to the differences in semantic approaches was the varying intellectual backgrounds of the practitioners involved. At the 2004 *Mathematical Foundations of Programming Semantics* confer- ence, it was mentioned that the panel consisted of an industrial worker (Jones), a mathematician (McCarthy), a physicist (Reynolds), and a lo- gician (Scott). These different intellectual backgrounds brought, along with the tools and techniques, research agendas—as pointed out by Ma- honey [Mah02]. Mathematicians, like McCarthy, tended to bring notions of formalising foundations and developing theories; that said, Strachey, who shared that desire, was not a trained mathematician at all (as he was happy to admit). Both Landin [Lan01] and Scott [Sco16] indepen- dently noted that training in universal algebra and lambda calculus had prepared them perfectly for mathematical views of programming. Van Wijngaarden's pragmatic approach to ALGOL 68 could be seen as the re- sult of his mathematical engineering training. And in contrast, one critic

of formal semantics, Saul Gorn, prided himself on being a practical programmer and preferred to think of language meaning in terms of machine functions [Gor64].

One way to gain insight into the way practitioners think about their work is suggested by Mahoney [Mah88]: look at the historical models they choose as their framing devices when presenting their own work (but remain critical). Here we can see McCarthy in [McC61] referring to Kepler's laws of planetary motion being derivable from Newton's laws of general motion as an example of a presented pedigree. Also notable is the way in which Strachey and his collaborators presented their 'mathematical' semantics, with heavy use of existing mathematical concepts such as lambda calculus and lattice theory. Indeed, the very name 'mathematical semantics' was chosen to not-so-subtly hint that other styles of semantics were insufficiently mathematical—Stoy admitted such in an interview [Sto16b].

Although there were significant differences in the intellectual backgrounds of the researchers in formal semantics, there was one major shared factor: recent experience with working on programming languages. McCarthy had been developing LISP [McC60] and the formulation (S-expressions) he had devised for this purpose clearly shines through in his early work on language description. Strachey had come to work on semantics from the painful experience of trying to develop a compiler for CPL, and realising that a shared intuition about the meanings of pro- gram constructs did not suffice to build a precise compiler. Landin had worked with Strachey creating an autocode for a Ferranti Orion com- puter prior to starting work on formal semantics. The researchers of the Vienna Group had, as one of their last tasks at their Technical University of Vienna (TUW) home before moving to IBM, developed an ALGOL 60 compiler for the Mailüfterl computer. Bekič and Lucas, the key intellec- tual drivers behind the project, recorded their experience in a technical report [LB62]; in this, they argue that without a "complete and unambiguous" formal definition, the task was long and difficult.

One notable exception is the creation of structural operational semantics by Gordon Plotkin, who had not been working on programming languages but rather on various aspects of language theory; however, he was putting together his approach in 1979, at which point the field was more advanced than for the majority of workers considered in this paper.

These experiences with programming languages, and the choices of language used to demonstrated the semantic techniques, made a noticeable difference to the formalisms. One obvious example is discussed in Section 7: the bigger the target language, the more complex the formal description needed to handle it. Some of the semanticists who tried to illustrate that their approach could work on full-scale languages really suffered from this. It is also worth observing that very many of the practitioners discussed herein used ALGOL 60 as a way to demonstrate their defini- tion technique: there are definitions in Landin's style [Lan65a; Lan65b], in VDL [Lau68], in a variant 'functional' style halfway between VDL and VDM [ACJ72], in denotational semantics [Mos74], and in VDM [HJ78] (as well as others). Writing a definition of ALGOL showed that the description technique could cope with the challenges of a full-scale language and its familiar object allowed better understanding of the definitions.

The writing of formal descriptions of ALGOL is covered in more detail in [AJ18].

The differing backgrounds of the practitioners involved, and their consequent differing motivations (see Section 4) led to variety in the applications of the formal descriptions produced. The Vienna Group frequently worked with PL/I, and had the notion of compilers for that language at the heart of most of their work. For Strachey, the formal description was essentially the end product. His collaborators, however, especially Peter Mosses and Robert Milne, found ways to apply denotational semantics to more practical tasks: a system for generating compilers from formal descriptions from the former [Mos75]; and a process for verifying implementations of languages based on hierarchical strata of refined descriptions from the latter [MS76]. Other workers, such as Landin and van Wijngaarden, applied their ideas of language description to language design. Landin's paper 'The Next 700 Programming Languages' [Lan66a] was heavily based on his "Imperative Applicative Expressions", a notion from his definition work, and was very influential on the later functional programming community.[17] Van Wijngaarden's language description technique grew from a sketched idea in the early 1960s [Wij62] to use in the full-blown book-length official definition of ALGOL 68 [WMPK68].

These different notions and agendas affected the choice of fundamental basis for the semantic descriptions. Strachey's preference for functions has been mentioned already; but his partnership with logician Scott led to the functions being given a very firm basis in mathematical (domain) theory. In contrast, McCarthy's use of functions was rather more like the loosely defined recursive expressions in his LISP work. While his semantics publications (e.g. [McC66]) used λ symbols to write functions, it was really only the notation that was borrowed from the calculus. McCarthy later admitted (in [McC80]) that he had the binding rules quite wrong and did not really understand Church.

Choices of base affected the language semantics in pervasive ways: as Strachey commented at an IFIP WG 2.2 meeting, assignment was hard in lambda calculus, but recursion tricky in others—despite both being used frequently in everyday programming languages [Wal69]. Klaus Samelson observed [in Lan66b] that Landin's decision to base his description on a lambda calculus framework meant increasing contortion in order to handle all the aspects of ALGOL 60.

As well as in basis, language description techniques varied hugely in their choice of notation. While notational differences were frequently written off as mostly unimportant in publications, many arguments between academics were over details of the symbols used to represent ideas. This might seem somewhat superficial, but the notation could make significant difference to the readability of description. Compare, for example, two denotational semantic descriptions of ALGOL 60: one written by Mosses of the PRG [Mos74], and the other in the VDM style of the VAB [HJ82]. An example is shown in Figure 1. The two use a very similar semantic approach[18] but appear on the surface totally different. Mosses uses a for-

[17] See commentary on this from e.g. Danvy [Dan09].

[18] There are some differences; see [AJ18] for more detail.

```
case"if Exp then Sta1 else Sta2":
   ℛ⟦Exp⟧ρ"boolean" {λβ. β → 𝔖⟦Sta1⟧ρθ, 𝔖⟦Sta2⟧ρθ}
```

```
M[mk-Condstmt(e,th,el)](stmtenv) ≜
    let (,env,cas) = stmtenv in
    def b          : M[e](env,cas);
    if b then M[s-sp(th)](stmtenv) else M[s-sp(el)](stmtenv)
```

Figure 1: Two denotational semantics descriptions of the same construct: a conditional statement. Above: Mosses [Mos74]; below: VDM [HJ82].

malised notation based on the Oxford fondness for single character names (often Greek letters) and which is intended to be machine-readable. Henhapl and Jones favour longer, multi-word identifiers and a looser syntax for better human reading. And, as discussed further in Section 7, the ALGOL 68 specification attracted serious criticism for its notation.

6 The importance of collaboration

The emphasis in the previous Section is on differences betweenpractitioners; however, there were also significantly similarities, especially within groups. Indeed, collaborative working was critical to the success of formal semantic techniques.

Almost all the researchers mentioned in this paper worked in groups. The IBM Laboratory had Bekič and Lucas as the main intellectual contributors [Neu16] but all the major documents carried lengthy author lists. The first version of their PL/I definition, for example, gives eight names [ULD66]. Despite the huge controversy it caused in IFIP's WG 2.1, the ALGOL 68 effort was nonetheless achieved by a group; although van Wijngaarden the "party ideologist" [Lin93] did rule resolutely. Łukasziewicz noted "Aad [van Wijngaarden] would rather go to the stake than renounce his principles" [Łuk85]. In Oxford, Strachey carefully assembled a small team of DPhil students and post-docs who shared his views on semantics. Many of the important breakthroughs that added extra depth to the denotational semantics method came from these: continuations, for example, were contributed largely due to the work of Chris Wadsworth. And while Plotkin was the sole author of SOS, he acknowledged the importance of working in the Edinburgh research environment where everyone was frequently sharing their ideas; the influences of Milner and Burstall were particularly valuable [Plo18].

One particularly important form of collaboration was the academic visit. A notable example is the travel of Dana Scott. In autumn 1969, he spent a term collaborating with Strachey that provided firm founda- tions for Strachey's semantics. This period was personally significant, too: Scott referred to it as "one of the best experiences in [his] personal life" [Sco77, p. 637]). However, he came to Oxford after a period of two years' sabbatical at Mathematisch Centrum in Amsterdam, where he had

been working with Jaco de Bakker. De Bakker, for his part, had begun to break away from van Wijngaarden's approach to language definition and collaborating with Scott led to a new 'Theory of Programs' [BS69]. Following his spell in Amsterdam, Scott spent the summer at the IBM Lab in Vienna, where he had been invited to help them understand Floyd's work on assertions [Flo67]. Instead, Scott spent the time telling them about his work with de Bakker, which planted the intellectual seeds for a later uptake of denotational semantics in Vienna. In 1968, Bekič of the Lab had also spent time on sabbatical: working with Landin in London, which also contributed to the Lab's later switch.

Landin himself had first developed his semantic technique while working for Strachey's consulting business in London in the early 1960s. Strachey wrote of this period: "It is an interesting comment on the state of the subject [programming language theory] that this work which at the time was probably the only work of its sort being carried out anywhere (certainly anywhere in England) was being financed privately by me." [Str71a], also quoted by [CK85]. It also seems likely that Landin's influence at this time was how Strachey became interested in lambda calculus.

The different institutions involved influenced the working practices of the researchers on formal semantics. IBM provided generous funding to the Vienna Lab for a long time, but the group was expensive—in part because Zemanek believed in employing almost as many support staff as he did scientists and himself had two secretaries.[19] He even managed to get IBM to pay for some IFIP business, including securing their sponsorship for the FLDL conference [Utm64b], and the bankrolling of two employees (a lawyer and an economist) to assist him as IFIP president [ZA87].

Zemanek preferred the industrial research setting and even turned down a request from the TUW to found a school of informatics because he felt he had more freedom to set the scope and direction of research at IBM [ZA87]. However, the intellectual goals of the group often clashed with the IBM management's desire for a product. In 1975, the Future Systems project for which the VAB had been developing a PL/I compiler based on a formal definition was unexpectedly cancelled. All staff had to find new work, and only a rather frantic struggle led to the publication of an edited book with their work from the period [BJ78]. Another example of the tension with IBM management was Language Control's horrified response to the 1966 PL/I definition, who were severely put off by the document's size and complexity. One commented that trying to use it to handle PL/I was like being asked to read *Principia Mathematica* [RW12] to perform addition [BBHM67].

In contrast, Strachey had struggled throughout the early 1960s to get financial support for academic programming language theory work. Even once the PRG was established in Oxford, the funding landscape was rather uncertain [CK85]. However, he had a lot of freedom to set the intel- lectual direction and was able to create what he described as "a highly critical and thoughtful atmosphere in which *ad hoc* or superficial ideas [were] given very short shrift" [Str71b]. Similarly, Plotkin argued that the wealthy status of Edinburgh University in the late 1970s and 1980s al-

[19] From personal communication with Kurt Walk, 2016.

lowed the existence of many small research groups with their own focuses, and that the best results came from interactions between these different groups [Plo18].

Beyond research groups, a number of important events brought to- gether practitioners in language description, especially a trio of confer- ences in the first half of the 1960s. The 1963 *Working Conference on Mechanical Language Structures* was held in Princeton. Its focus was not on semantics, but the discussion reported by Gorn [Gor64] shows that many semantic concerns were discussed. The second, *Formal Language Description Languages*, in Baden-bei-Wien in 1964, had many important early papers on semantics. A third conference, on *Programming Lan- guages and Pragmatics*, was held in 1965 in San Dimas, California. This trio of conferences was seen by attendees as corresponding to the Morris separation of linguistics into syntax, semantics, and pragmatics, respec- tively. These conferences served to codify the field of formal language description and establish a research agenda: namely, that programming languages were worthy objects of study, and that formalism provided the toolkit for that study.

One important result of the FLDL conference was the creation of IFIP Working Group 2.2, on formal language description languages (the name was subsequently changed a number of times). Ideas for a working group on this topic had been floated at the parent committee, TC-2, from as early as 1962, but it was the success of the 1964 conference that got them going in earnest. The Working Group was founded, in 1967, to bring together everyone working in the field for discussion and interaction. Al- most from the very start, it was marred by bickering and fighting between researchers proposing different approaches, often over fairly trivial details of presentations. It is almost saddening to read the minutes of Working Group 2.2 and see how the negative reactions to each others' presentations caused such disunity. Indeed, such an atmosphere may have contributed to the extremist positions taken by a number of members.

The working group also suffered from frequent crises of identity, and arguments over terminology, as well as the best way to run group meetings. A typical excerpt from the minutes of the second meeting runs "There was a discussion on how best to proceed with the discussion" [Wal68]. These arguments over scope and purpose of research demonstrate Mahoney's concept of agenda setting as the point at which a field becomes established. A similar experience had been felt by WG 2.1 a few years prior during the creation of ALGOL 68. Indeed, the fallout from that attempt to produce a unified product and the division created instead may have been a factor in preventing WG 2.2 from trying to achieve something cohesive as a group. Following a series of meetings intended to find a purpose for the group, the sixth meeting, in 1971, was cancelled after only a few members showed up and went on hiatus while TC-2 tried to decide how to handle the situation [Pec71]. Eventually, the working group reformed in 1974 under a new chair, Erich Neuhold, and with a much broader remit: formal description of programming concepts [Dub73].

All of this did not mean the group had been a complete failure: it had achieved remarkable success in bringing people together. The third meeting, in April 1969, was particularly noteworthy, as it was where Stra-

chey and Scott met for the first time and their plans to work together began. It was also where Hoare's axiomatic method saw its first public airing [Wal69].

There is an interesting comparison to be made with the Nicholas Bourbaki group of French mathematicians in the middle of the 20th Century.[20] Bourbaki had attempted to fix the foundations of mathematics by committee but suffered infighting from the strong-held but different beliefs of its members. Indeed, the comparison had even occurred to members of WG 2.2: at their second meeting, Tabory remarked "Bourbaki builds a new cathedral out of well-known things and gives convenient language for these things. It is important that we are open-minded for new things which are not as yet in the cathedral" [Wal69]. Sadly, however, there was not a great deal of open-mindedness amongst attendees of the meetings, and new proposals more frequently met with criticism.

The experience of WG 2.2 demonstrates an unfortunate counter to the main thrust of this Section: while collaborations were indeed essential for success in formal semantics work, influence tended only to exist within ideological groups. One counter to this is the early work of McCarthy and Landin who contributed ideas that ran through a number of different later approaches. Another important story is the experience of the Vienna Lab. By the end of the 1960s, they were uncovering severe technical problems with their VDL approach, and work on that slowed. At the beginning of the 1970s, however, the group began to experiment with a totally new style. They fused the core of Strachey's denotational semantics with ideas from earlier Vienna work, creating the VDM style. This represents a rare example of semantics researchers picking up techniques from a different approach, and was made possible due to Scott's and Bekič's visits mentioned earlier. Jones, who rejoined the Vienna Lab at this point, had also attended Strachey's seminars in Oxford.

These examples aside, idea sharing across fundamentally different bases was rare, and criticism was the norm. But this is in many ways indicative of the progress of science generally: there are no lone geniuses, but rather a mass of arguing and critical colleagues.

7 Criticism

The previous Section closed with a mention of criticism, and formal semantics faced a great deal of that—both from other practitioners with different styles, and from interested outsiders.

The most common criticism levelled at formal semantics descriptions is that they were far too large and complex. Frequently, this was due to efforts to model the entirety of a programming language. Small examples in each of the semantic styles showed that it was relatively easy to give semantics to trivial chunks of programs, but at the scale of large languages, semantic descriptions became very unwieldy. This distinction between complexity in definition method and language under definition was of- ten overlooked. Although van Wijngaarden's and the VAB's description

[20] The author thanks Chris Hollings for this suggestion.

techniques were hardly slimline, a lot of the bulk in the definitions of ALGOL 68 and PL/I (respectively) was due to richness of features in those languages.[21] Strachey and Milne addressed this very problem in the introduction to their essay submitted for the Cambridge Adams Prize 1973–4: "A superficial glance will show that our essay is long and our notation elaborate. The basic reason for this unwelcome fact is that programming languages are themselves large and complex objects which introduce many subtle and rather unfamiliar concepts." [MS74, p. 22]

An unfortunate corollary to this criticism is that semanticists were often also attacked for focusing too much on their metalanguage than the language under definition. IBM's Language Control Board were frustrated that the Vienna Group appeared to have been more interested in creating a general-purpose definition method than making something useful for PL/I. Similar comments were made to van Wijngaarden, suggesting that he was more concerned with his two-level grammars than with ALGOL 68. So, these people were attacked for not focusing enough on the object languages; but also criticised when their descriptions displayed complexity resulting from careful attention to all the languages' features.

This leads to a more general concern that was raised during discussions of formal semantics, which the present author terms "expressiveness versus elegance".[22] Bringing back to mind the quotation from Hoare that opens this paper, the question was: should a description be small, easy to read, and ultimately fairly simple; or should it be large, powerful, and (perhaps necessarily) clunky? This argument came up at Working Group 2.2 [Wal69]:

> Caracciolo: A reduction to simpler questions would mean to omit the proper problem.
>
> Scott: Only the most primitive, non-problematic things have been dealt with using this approach.
>
> Laski: A language definition should specify as little as possible.

Here we see Laski embodying the 'elegance' side, particularly beloved by Hoare (it is his axiomatic method they are discussing) and Dijkstra; where Scott and Caracciolo seem to be favouring 'expressiveness'. It was noted by Wadsworth that Strachey tended to prefer modelling smaller languages to show off the proof of concept [Wad00] although by the mid 1970s, he and Milne had chosen to include a full-size language's definition in their Adams Essay and subsequent book [MS76].

Another reason semantics practitioners struggled to find early acceptance among certain of their peers was the intellectual atmosphere of the

[21] This is illustrated by the following exchange, taken from the minutes of a WG 2.1 meeting [Tur67]:

TURSKI: In Grenoble we decided that the proposed description method is a milestone in the development of the language.

RANDELL: A milestone or a millstone?

General laughter follows.

[22] The expressiveness–elegance dichotomy may reflect similar trends in programming, computing, mathematics, and perhaps even science as a whole. Tedre mentions briefly a clash between practical efficiency and scientific purity, which fits this model [Ted14, p. 114]. This is an area that seems ripe for further exploration.

1960s. Peter Wegner called the 1950s an "age of empirical discovery" in computer science [Weg76] and although this was changing in the 1960s, as Tedre [Ted14] points out, there were plenty of people in the early 1960s who still thought of computing and computation in terms of machines. One such was Saul Gorn, who said at the WCMLS in 1963 "I am one of those extremists who feel that it is impossible to separate a language from its interpreting machine" [Gor64]. Another person with similar views was Klaus Samelson, who said to McCarthy "I would rather avoid the word "semantics" altogether. Semantics is what we have in our heads; as soon as we write it down it's not semantics any more" [in McC66].

A further criticism against formalising meaning was that it essentially meant that programmers had to learn another language. John Backus, who had worked on FORTRAN as well as developing a formal notation for the syntax of programming languages, described formal language defi- nitions in a later interview as "pages and pages of gobbledygook" through which programmers were expected to wade [SB79]. McCarthy, however, had anticipated such criticism from the beginning, writing in his FLDL paper "I have written down this notation, and unless it is substantially simpler in some intuitive way than ALGOL, you can say, 'What has been gained?' I have merely given you a new language to learn" [McC66]. He continued by noting that Tarski had faced similar arguments when writing his semantics of logic, and yet that had proven useful; and furthermore a carefully chosen formalism should be simpler than its object language. The metalanguage could use a smaller set of base objects, familiar to mathematicians, and the notation could be explained with careful natural language.

An unusual take on this problem was provided by de Bakker, whose definition of ALGOL 60 [Bak67] used a somewhat circular approach. The interpreter was written in ALGOL and so the reader was expected to gain some intuitive understanding of its function first, then use that to get to grips with ALGOL, and finally return to the interpreter and understand it fully.

A final criticism of formal semantics was ultimately philosophical in nature. This has become termed the 'materiality argument' and is largely due to Fetzer [Fet88]. He criticised formal modelling *per se*, arguing that formal proofs are inherently flawed because they deal with abstract objects and yet claim to say things about real-word artefacts. Although this charge was not directly levelled at formal semantics, there is a clear connection between verification and semantics. Even Zemanek brought up this problem, writing "No formalism makes any sense in itself; no formal structure has a meaning unless it is related to an informal environment [...] the beginning and the end of every task in the real world is infor- mal" [Zem80].

Another element to this concern was that proofs should be social, rather than formal, processes: this was the core thesis in a paper by DeMillo, Lipton, and Perlis [DLP79]. The argument was that fully formal proofs are too large, unintuitive, and error-prone—all criticisms made of formal semantics too. This 'verification debate' was at times rather

acrimonious;[23] accounts differ on just how influential it was on formal computer science. [Mac01, Ch. 6], [Ted14, Ch. 4], [Pri19, § 7.3] and [Ede07] all cover the argument, and it remains a fierce area of discussion in philosophy of computing.[24]

As mentioned, this debate focused largely on formal verification, and, indeed, some semanticists saw their work as a partial answer to the problem of trying to link the abstract and physical. One of the central prob- lems in computing is to determine whether the machine is doing what the programmer wants. As this is a difficult task, it is easier split into parts, as Joe Stoy argued in an interview [Sto16b]. He discussed the value of a "nested set of confidence-inspiring procedures" with a high-level program at one end and machine actions at the other—and a crucial spot in the middle for the formal semantics of the programming language. Formal verificationist J Moore accomplished this in an industrial setting in the late 1980s, explaining that his company had a "computational logic stack" of programs, compilers, operating systems, and so on. Each was proven to be a correct implementation of the previous level [Moo17]. Moore noted that semantics was a crucial part of this: "semantics [of the languages used] is something I have to have in order to get my formulas".

Ultimately, Stoy argued that formal semantics did not achieve the successes hoped for because of these problems, and because people were managing fine with natural language definitions [Sto16b]. Plotkin agreed with this, noting that real success would have required close collaboration between semantics workers and language designers, which was difficult to achieve [Plo18]. Mosses noted one more prosaic problem: computer users today are used to benefiting from tool support, and there are very few available for formal semantics [Mos01].

8 Building a European computing theory

Despite the many criticisms levelled at formal semantics, it had a cru- cial role in the formation of theoretical computer science, particularly in Europe. In 1976, Peter Wegner wrote a paper about trends in research paradigms in computing [Weg76]. Wegner was a one time member of IFIP WG 2.2 with an interest in programming language theory—indeed, he had written a paper about the Vienna Lab's early language defini- tion work [Weg72]—and he also had written a an earlier paper about the disciplinary identity of computing [Weg70]. In his paper on research paradigms, he proposed that computing research has three major aspects: scientific, mathematical, and technological. Although Wegner was only halfway into the 1970s at time of his writing, he made a compelling case for computing research (specifically *programming* research—there is little material in this paper on, for example, hardware) going through different phases of focus corresponding to decades. He noted that the 1950s saw

[23] Glass suggests that the debate was so unpleasant because the computer science field had yet to sufficiently mature to allow room for dissenting voices and differing agendas [Gla02].

[24] See, for example, the (self-published) paper by Daylight [DBF16].

much work on discovering and implementing core aspects of programming, such as compilers, libraries of routines, and link loaders. In the 1960s, work focused on building theories of programming that captured abstract and fundamental properties of these systems. Clearly, the work on programming language semantics reported in this paper fits firmly into this category. The 1970s, and possibly onwards, Wegner argued, brought a more practical direction, with research on turning the theoretical work towards applications such as verification systems.

Matti Tedre, whose 2014 book explores the disciplinary identity of computing (and how it changed) in considerable detail, roughly tends to agree with Wegner's three categorisations, and divides his book into sections along these lines. However, Tedre's work is more nuanced, and discusses the engineering, scientific, and mathematical traditions of computing that continued throughout the entirety of the 20th Century. The semantics work discussed in the present paper mostly fits in the mathematical categorisation, as has already been discussed in some detail. However, it is also worth briefly looking at the scientific aspects. According to Tedre, computer science in the 1960s and into the 1970s tended to be expressed in position papers where an author put forward their ideas about how various aspects of computing worked, without the kind of experimental validation so central to most other scientific fields [Ted14, Ch. 9]. This began to come under fire in the 1980s and later, but in the 60s was very common. Considering the arguments over theories of programming at WG 2.2 meetings discussed above, and the almost polemic quality to many of the semantics publications, this fits within the academic norm of the time.

Formal semantics of programming languages was a particularly crucial part of this 1960s vogue for theorising: Mahoney placed semantics as one of the three main pillars of theoretical computing[25] in the period from 1955 to 1975 [Mah02, pp. 28–9]. The other two pillars, automata and formal languages, and complexity theory, have their own different historical paths; but is remarkable to observe that what now seems like a very large corpus of academic theory has grown from essentially these three roots.

An interesting aspect to this division of computing theory is that there was a geographical element: very many of the researchers discussed in the present paper are European, which is reflective of the general trend at the time. Indeed, Plotkin recalled, in an interview, delivering a talk on semantics at a large American conference on theoretical computing and attendance being so low that semantics was removed from future conference programs [Plo18]. Mahoney commented on this geographical divide, noting that in the 1970s there were "fundamental differences between the formal, mathematical orientation of European computer scientists and the practical, industrial focus of their American counterparts" [Mah88, p. 123].

[25]Mahoney's precise term is 'theoretical computer science' and he has the following to say about that: "It is curious that to this day the community distinguishes between computer science and theoretical computer science, as if the former involves some kind of science other than theoretical science. It is not clear what that other kind of science might be nor what is scientific about it" [Mah02, p. 28].

J Moore, who had completed an undergraduate degree at MIT and came to Edinburgh in the early 1970s for a PhD, felt a stark difference in the attitude towards computers: "It was the difference between a resource- rich engineering environment, very experimental, in America—or at least at MIT—and [at Edinburgh] very theoretical, treating the computer as this delicate, holy object" [Moo17]. At MIT, Moore had been used to being able to play around and experiment with things on the computer, not worrying if something didn't work right away, but in Edinburgh time on the machines was too precious not to have everything already worked out in advance. Moore suggested that this could have been for financial reasons: Europe in the 1960s was still feeling the after-effects of the Second World War and could not afford to have so many computing machines. This could have played a part in the growth of a European attitude to computing that emphasised lengthy intellectual pen-and-paper work prior to sitting in front of a terminal. Indeed, talk to any person who had been computing in the 1960s or 70s in Europe and they remember how precious a resource computer time still was—stories abound of the pain of waiting days for your program to be run only to have it returned by the computer techs telling you it had crashed.

However, this does not seem a sufficient explanation for the concentrating of work on formal semantics in Europe. It may have played a part, but it seems likely that the major factor was the intellectual history discussed in Section 5, and the other motivators in Section 4. Another suggestion was provided in an interview with Plotkin, in which he simply stated that Americans were more pragmatic about programming, having commercialised the computer sooner, and were not likely to spend time theorising about it [Plo18].

9 Impacts of semantics

Formal semantics became a central part of theoretical computing as computer science solidified into its own discipline, but did not break through into mainstream mathematics [Ted14, Ch. 3]. Tedre explains that this was because mathematicians had different research goals, and tended to see computing as much too *applied*—especially as the maths in computing was often finite rather than infinite. Lack of impact in core mathematics does not, however, mean that semantics work was without influence.

Through the late 1970s and into the 1980s a trend was emerging: researchers who had been working on formal semantics were changing their focus. The major problem had been that trying to define whole programming languages meant working with very large and unwieldy objects. There was instead a growing recognition that the tasks they were attempting to use formal semantics for—such as determining whether a program was working correctly—could be decomposed into smaller problems. That was the approach followed by the Vienna Lab in the mid-1970s, as they turned their formalism from a language definition tool into one for specifying individual programs. This also fits with Wegner's notion of the 1970s as a time in which attention turned towards more practically use- ful applications of computing theory [Weg76]. The other criticisms listed

previously in Section 7 may have played a part in dissuading researchers from working on programming language definition. However, this should not be taken to mean that the work was without impact; on the contrary, principles and approaches developed for formal semantics continued to have great influence on many angles of theoretical computing.

Formal techniques for proving things about programs, or 'formal methods' as they became known, grew from the work of Hoare on axiomatic semantics, the Vienna VDM approach, and work in Edinburgh on theorem proving. Program specification and verification was an avenue developed especially from VDM, and new families of specification languages grew up. One amusing reflection of earlier work is that these specification languages themselves were sometimes given formal definitions: examples include Clear [BG80] and Z [Spi88]. For a lengthier treatment of the history of program verification, see [Jon03].

One area that saw particular fruitful research was on concurrency: through the late 70s and 80s many theories of parallel programming grew up that borrowed terminology and concepts from language definition. Amir Pnueli, who alongside others developed a formalism for concurrency which became known as temporal logic, talked about the 'semantics' of concurrent programs formalised in his logic [Pnu79]. Rather than trying to formalise an entire language, however, this work moved in the direction of formalising the particular temporal properties which were of interest for concurrency problems. This was the general trend in concurrency research: initial ideas came from semantics or program verification work, but isolating and working purely on problems caused by concurrency was a tough enough task itself.

One of Strachey's focuses towards the end of his life,[26] and especially during his collaborative work with Robert Milne, was the idea of a 'semantic bridge'. This involved developing multiple definitions of the same programming language with different bases or refinements, to move an abstract language concept closer to an implementation model. This was influential on a later research direction championed by Hoare: the idea of 'unifying theories of programming'. Hoare first work in this area was with Peter Lauer, who had written the VDL definition of ALGOL 60, and later went to Belfast to complete a PhD with Hoare. Together, they worked on a method to use a semantic model to verify the consistency of an axiomatic system [HL74]. Later, while Hoare was head of the PRG in Oxford in the 1980s, he saw a number of people working on different theories of programming, and hoped to find a way to bring them together [He18]. This rather grandiose task resulted ultimately in a book co-authored with He Jifeng [HH98] where the hope was to unify computing theories just as algebra had been unified.

Another important area of work growing from language definition is type theory, which took a lot of influence from the domains developed by Scott for denotational semantics [Sch00]. Context conditions, as used in the ALGOL 68 definition and later in VDM, also offered a structured and powerful way to think about typing and properties of constructs. Rich typing systems give programmers the ability to strongly shape their programs

[26] Strachey died suddenly in 1975, less than 60 years old, from liver disease.

around user-defined types, and this is one area of formal computing that has managed to break through into mainstream programming. A good historical treatment of type theory, particularly its origins in logic, can be found in [CH06].

The world of functional programming also owes a great deal to for- mal semantics, especially denotational semantics and the early work of Landin. Many important principles in this paradigm, such as functions as first-class citizens, closures, and continuations come from these sources. Indeed, David Turner, who was a DPhil student under Strachey, said he owed a great deal to that period at the PRG: "One way I've had an understanding of what I'm doing with languages like SASL [his first func- tional programming language], is that I was turning the meta-language of denotational semantics into a programming language." [Tur19]

A final and perhaps most important impact of formal semantics work, and all the work on programming language theory of the 1960s in gen- eral, is that it opened up a whole new area of research. Programming languages became objects worthy of study, dissection, and analysis, in- dependent of their role as vehicles for programs. Mark Priestley argued that this period represented a paradigm shift in computer science and that it had been sparked by the machine-independent definition of AL- GOL 60 [Pri11, § 9.3]. Formal semantics provided the tools and techniques for examining and constructing languages in this new way [Sch00], and so deserves recognition as one of the important foundational steps in com- puter science.

10 Conclusions

The introduction of high level programming languages brought great challenges to computing, and getting to grips with these complex objects required a lot of effort as well as new ideas not previously seen in mathematics and logic. The costs (in terms of time and money) associated with getting programming languages and their compilers right were huge, and addressing this using formalism was an important activity in the 1960s and through into the 1980s. Formal syntax was a clear success story with great applicability in parsers and compilers but semantics was a much bigger challenge. While it is not the case that, as the early semanticists had hoped, every programming language has a full formal definition, work on semantics impacted in a number of important technical areas.

Formalising the semantics of programming languages was always a difficult task, and dealing with complex languages packed with features only made it harder. Despite this, researchers rose to the challenge to address various goals: finding a theory of computing, figuring out the capabilities of computers, getting to grips with programming languages and compilers, and aiding reasoning about programs.

Work on semantics developed in different ways in different places, particularly due to the intellectual backgrounds of the researchers involved. One common experience for nearly all the researchers mentioned was working with programming languages, particularly design and implementation, and encountering a great deal of difficulty. Similarities

existed across styles, but there were differences—both superficial and fundamental—that had real impact on power and usability of the formalisms. Various groups formed to work on the topic, and academic visitations played an important role in spreading ideas around during the early period. Later work, however, often crystallised into dogmas around different approaches to semantics, and influence across styles was uncommon. The institutions in which the researchers operated also tended to affect the goals and outcomes of the research.

Assorted criticisms were levelled at formal semantics work, most commonly that the language descriptions produced were too large and complex to be usefully understood. The need to understand the metalanguage being used was often seen as a barrier to their widespread adoption. More esoteric criticisms included a tension between the desire for expressivity or elegance in the formalisms, and a philosophical debate about the reliability and utility of models in determining real-world behaviour.

Despite these criticisms, formal semantics can be seen as really starting off the idea of formalised theoretical computing—at least in the European sense. Research on formal semantics had particular impact in the areas of program verification, type theory, and programming language theory and design. This indicates that theoretical computing is a rich and fertile area for further historical research.

References

[ACJ72] C. D. Allen, D. N. Chapman, and C. B. Jones. *A For- mal Definition of ALGOL 60*. Tech. rep. 12.105. IBM Laboratory Hursley, 1972. url: http://homepages.cs.ncl.ac.uk/cliff.jones/ftp-stuff/TR12.105.pdf.

[AJ18] Troy K. Astarte and Cliff B. Jones. "Formal Semantics of ALGOL 60: Four Descriptions in their Historical Context". In: *Reflections on Programming Systems - Historical and Philosophical Aspects*. Ed. by Liesbeth De Mol and Giuseppe Primiero. Springer Philosophical Studies Series, 2018, pp. 71–141.

[Ast19] Troy K. Astarte. "Formalising Meaning: a History of Programming Language Semantics". PhD thesis. Newcastle University, 2019.

[Bac54] John W. Backus. "The IBM 701 speedcoding system". In: *Journal of the ACM* 1.1 (1954), pp. 4–6.

[Bac59] John Warner Backus. "The Syntax and Semantics of the Proposed International Algebraic Language of the Zurich ACM-GAMM Conference". In: *Proceedings of the International Conference on Information Processing*. UNESCO. 1959, pp. 125–132.

[Bak67] Jacobus Willem de Bakker. *Formal definition of programming languages. With an application to the definition of ALGOL 60*. Mathematical Centre tracts: 16. Amsterdam: Mathematisch Centrum, 1967.

[Bau72] Friedrich L. Bauer. "From Scientific Computation to Computer Science". In: *The Skyline of Information Processing: Proceedings of the tenth anniversary celebration of the IFIP*. Ed. by Heinz Zemanek. IFIP. North-Holland, 1972.

[BBHM67] G. W. Bonsall, M. R. Barnett, R. M. Harrington, and H. Morrison. *ULD3 and Language Development*. IBM internal memo to J. L. Cox. Technical University of Vienna NL. 14. 072/2 Zemanek. PL/I History Documents. 1967.

[BG80] Rod M. Burstall and Joseph A. Goguen. "The semantics of Clear, a specification language". In: *Abstract Software Specifications*. Lecture Notes in Computer Science 86. Springer. 1980, pp. 292–332.

[BH14] Dines Bjørner and Klaus Havelund. "40 years of formal methods". In: *International Symposium on Formal Methods*. Springer. 2014, pp. 42–61.

[BJ78] D. Bjørner and C. B. Jones, eds. *The Vienna Development Method: The Meta-Language*. Vol. 61. LNCS. Springer-Verlag, 1978. isbn: 978-3-540-08766-3. doi: 10.1007/3-540-08766-4. url: http://dx.doi.org/10.1007/3-540-08766-4.

[BS69] J. W. de Bakker and D. Scott. "A Theory of Programs". Manuscript notes for IBM Seminar, Vienna. 1969.

[CH06] Felice Cardone and J Roger Hindley. "History of lambda-calculus and combinatory logic". In: *Handbook of the History of Logic* 5 (2006), pp. 723–817.

[Cho56] Noam Chomsky. "Three models for the description of language". In: *IRE Transactions on information theory* 2.3 (1956), pp. 113–124.

[Chu41] A. Church. *The Calculi of Lambda-Conversion*. Princeton University Press, 1941.

[CK85] Martin Campbell-Kelly. "Christopher Strachey, 1916-1975: A Biographical Note". In: *IEEE Annals of the History of Computing* 1.7 (1985), pp. 19–42.

[CKA96] Martin Campbell-Kelly and William Aspray. *Computer: a history of the information machine*. The Sloan technology series. New York, N.Y: Basic Books, 1996. isbn: 0465029892.

[Dan09] Olivier Danvy. "Peter J. Landin (1930–2009)". In: *Higher-Order and Symbolic Computation* 22.2 (2009), pp. 191–195.

[DBF16] Edgar G Daylight, Raymond Boute, and Arthur C Fleck. *Turing Tales*. Lonely Scholar, 2016.

[Dij61] Edsger Wybe Dijkstra. *On the Design of Machine Independent Programming Languages*. Report MR 34. Mathematisch Centrum, 1961. url: http://www.cs.utexas.edu/users/EWD/transcriptions/MCreps/MR34.html.

[Dij72] E. W. Dijkstra. "The Humble Programmer". In: 15.10 (1972), pp. 859–866.

[Dij74] Edsger Wybe Dijkstra. *Letter to Hans Bekič*. Held in the Dijkstra archive online. EWD 454. 1974. url: http://www.cs.utexas.edu/users/EWD/transcriptions/EWD04xx/EWD454.html.

[DLP79] R. A. DeMillo, R. J. Lipton, and A. J. Perlis. "Social Processes and Proofs of Theorems and Programs". In: 22.5 (1979), pp. 271–280.

[Dub73] J. J. Duby. *Minutes of the policy subcommittee of Working Group 2.2*. Held in the Ershov archive (online). 1973. url: http://ershov.iis.nsk.su/en/node/806092.

[Ede07] Amnon H. Eden. "Three Paradigms of Computer Science". In: *Minds and Machines* 17.2 (2007), pp. 135–167. issn: 1572-8641. doi: 10.1007/s11023-007-9060-8. url: https://doi.org/10.1007/s11023-007-9060-8.

[Fee81] J. M. Feeney. "Management Information Systems - The Failure of Technology". In: *Business Information Systems*. 9 7. Pergamon/Infotech, 1981.

[Fet88] James H Fetzer. "Program verification: the very idea". In: *Communications of the ACM* 31.9 (1988), pp. 1048–1063.

[FF12] Alan Foley and Beth A. Ferri. "Technology for peo- ple, not disabilities: ensuring access and inclusion". In: *Journal of Research in Special Educational Needs* 12.4 (2012), pp. 192–200. doi: 10.1111/j.1471-3802.2011.01230.x. url: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1471-3802.2011.01230.x.

[Flo67] R. W. Floyd. "Assigning Meanings to Programs". In: *Proc. Symp. in Applied Mathematics, Vol.19: Mathematical Aspects of Computer Science*. American Mathematical Society, 1967, pp. 19–32.

[Gla02] Robert L Glass. “The proof of correctness wars”. In: *cacm* 45.8 (2002), pp. 19–21.

[Gle52] Alick E. Glennie. *The automatic coding of an electronic computer*. Talk delivered at Cambridge University, February 1953. 1952.

[Gor64] Saul Gorn. “Summary Remarks (to a Working Conference on Mechanical Language Structures)”. In: *Communications of the ACM* 7.2 (Feb. 1964), pp. 133–136. url: http://doi.acm.org/10.1145/363921.363946.

[Han07] Sven Ove Hansson. “The Ethics of Enabling Technology”. In: *Cambridge Quarterly of Healthcare Ethics* 16.3 (2007), pp. 257–267. doi: 10.1017/S0963180107070296.

[He18] Jifeng He. *Unifying theories of refinement*. Unified Theories of Programming 20th-anniversary BCS-FACS Evening Seminar. Introduction by Tony Hoare; summary by Jim Woodcock. Oct. 2018. url: https://www.bcs.org/content/ConWebDoc/59563.

[HH98] C. A. R. Hoare and Jifeng He. *Unifying Theories of Programming*. Prentice Hall, 1998.

[HJ78] Wolfgang Henhapl and Cliff B. Jones. “A Formal Definition of ALGOL 60 as Described in the 1975 Modified Report”. In: *The Vienna Development Method: The Meta-Language*. Ed. by D. Bjørner and Cliff B. Jones. Vol. 61. Lecture Notes in Computer Science. Springer-Verlag, 1978, pp. 305–336. doi: 10.1007/3-540-08766-4_12. url: http://homepages.cs.ncl.ac.uk/cliff.jones/publications/OCRd/HJ82.pdf.

[HJ82] Wolfgang Henhapl and Cliff B. Jones. “ALGOL 60”. In: *Formal Specification and Software Development*. Ed. by Dines Bjørner and Cliff B. Jones. Prentice Hall International, 1982. Chap. 6, pp. 141–174. url: http://homepages.cs.ncl.ac.uk/cliff.jones/ftp-stuff/BjornerJones1982/Chapter-6.pdf.

[HL74] C. A. R. Hoare and P. E. Lauer. “Consistent and complementary formal theories of the semantics of programming languages”. English. In: *Acta Informatica* 3.2 (1974), pp. 135–153. issn: 0001-5903. doi: 10.1007/BF00264034. url: http://dx.doi.org/10.1007/BF00264034.

[HMT87] Robert Harper, Robin Milner, and Mads Tofte. *The Semantics of Standard ML: Version 1*. Hard copy. University of Edinburgh, Department of Computer Science, Laboratory for Foundations of Computer Science, 1987.

[Hoa00] C. A. R. Hoare. “A Hard Act to Follow”. In: *Higher-Order and Symbolic Computation* 13.1 (2000), pp. 71–72.

[Hoa69] C. A. R. Hoare. “An axiomatic basis for computer programming”. In: *Communications of the ACM* 12.10 (1969), pp. 576–580.

[Hop81] Grace Murray Hopper. “Keynote address at ACM SIGPLAN History of Programming Languages conference, June C1–3 1978”. In: *[Wex81]*. Ed. by Richard L. Wexelblat. ACM Monograph Series. 1981.

[JA18] Cliff B. Jones and Troy K. Astarte. “Challenges for semantic description: comparing responses from the main approaches”. In: *Proceedings of the 3rd School on Engineering Trustworthy Software Systems*. Ed. by Jonathan P. Bowen and Zhiming Liu. Vol. 11174. Lecture Notes in Computer Science. 2018, pp. 176–217.

[JMSR04] Cliff B. Jones, John McCarthy, Dana S. Scott, and John C. Reynolds. *Reminiscences on Programming Languages and their Semantics*. Panel at Mathematical Foundations of Programming Language Semantics XX. 2004.

[Jon03] Cliff B. Jones. “The Early Search for Tractable Ways of Reasoning about Programs”. In: *IEEE, Annals of the History of Computing* 25.2 (2003), pp. 26–49. doi: 10.1109/MAHC.2003.1203057. url: http://doi.ieeecomputersociety.org/10.1109/MAHC.2003.1203057.

[Kri63] Saul A. Kripke. “Semantical Considerations on Modal and Intuitionistic Logic”. In: *Acta Philosophica Fennica* 16 (1963), pp. 83–94.

[Lan01] Peter J. Landin. “Reminiscences”. In: *Program Verification and Semantics: The Early Work*. A seminar held at the Science Museum, London. 2001. url: https://vimeo.com/8955127.

[Lan65a] Peter J. Landin. “A Correspondence Between ALGOL 60 and Church’s Lambda-notation: Part I”. In: *Communications of the ACM* 8.2 (Feb. 1965), pp. 89–101. url: http://doi.acm.org/10.1145/363744.363749.

[Lan65b] Peter J. Landin. “A Correspondence Between ALGOL 60 and Church’s Lambda-notation: Part II”. In: *Communications of the ACM* 8.3 (Mar. 1965), pp. 158–167. url: http://doi.acm.org/10.1145/363791.363804.

[Lan66a] P.J. Landin. “The next 700 Programming Languages”. In: *Communications of the ACM* 9 (1966), pp. 157–166.

[Lan66b] Peter J. Landin. “A formal description of ALGOL 60”. In: *Formal Language Description Languages for Computer Programming*. North-Holland, 1966, pp. 266–294.

[Lau68] Peter E. Lauer. *Formal definition of ALGOL 60*. Tech. rep. 25.088. IBM Laboratory Vienna, 1968. url: http://homepages.cs.ncl.ac.uk/cliff.jones/publications/OCRd/Lau68.pdf.

[LB62] Peter Lucas and Hans Bekič. *Compilation of ALGOL, Part I—Organization of the Object Program*. Laboratory report LR 25.3.001. IBM Laboratory Vienna, 1962.

[Lin93] C. H. Lindsey. “A History of ALGOL 68”. In: *The Second ACM SIGPLAN Conference on History of Programming Languages*. HOPL-II. ACM, 1993, pp. 97–132. doi: 10.1145/154766.155365.

[Luc78] Peter Lucas. “On the formalization of programming languages: Early history and main approaches”. In: *The Vienna Development Method: The Meta-Language*. Vol. 61. LNCS. Springer, 1978, pp. 1–23.

[Mac01] Donald MacKenzie. *Mechanizing Proof: Computing, Risk, and Trust*. MIT Press, 2001.

[Mah02] Michael Sean Mahoney. “Software as Science—Science as Software”. In: *History of Computing: Software Issues*. Ed. by Ulf Hashagen, Reinhard Keil-Slawik, and Arthur Norberg. Springer, 2002, pp. 25–48.

[Mah88] Michael S Mahoney. “The history of computing in the history of technology”. In: *Annals of the History of Computing* 10.2 (1988), pp. 113–125.

[Mah96] Michael S. Mahoney. “What Makes History?” In: *His- tory of Programming languages—II*. Ed. by Thomas J. Bergin Jr. and Richard G. Gibson Jr. New York, NY, USA: ACM, 1996, pp. 831–832. doi: 10.1145/234286.1057848. url: http://doi.acm.org/10.1145/234286.1057848.

[McC60] John McCarthy. “Recursive functions of symbolic expressions and their computation by machine, Part I”. In: *Communications of the ACM* 3.4 (1960), pp. 184–195.

[McC61] John McCarthy. “A Basis for a Mathematical Theory of Computation, Preliminary Report”. In: *Papers Presented at the May 9-11, 1961, Western Joint IRE-AIEE-ACM Computer Conference*. IRE-AIEE-ACM ’61 (Western). ACM, 1961, pp. 225–238. url: http://doi.acm.org/10.1145/1460690.1460715.

[McC63] John McCarthy. “Towards a Mathematical Science of Computation”. In: *IFIP Congress*. Vol. 62. 1963, pp. 21–28.

[McC66] John McCarthy. “A formal description of a subset of ALGOL”. In: *Formal Language Description Languages for Computer Programming*. North-Holland, 1966, pp. 1–12.

[McC80] John McCarthy. “LISP—notes on its past and future—1980”. In: *Proceedings of the 1980 ACM conference on LISP and functional programming*. ACM. 1980, pp. 5–viii.

[Mil93] Robin Milner. *Interviewed by Tony Dale*. Edinburgh, UK. 1993.

[MLB76] Michael Marcotty, Henry Ledgard, and Gregor V Bochmann. “A sampler of formal definitions”. In: *ACM Computing Surveys (CSUR)* 8.2 (1976), pp. 191–276.

[Moo17] J Strother Moore. *Interview with J Strother Moore*. Conducted by Troy Astarte. 2017.

[Mor46] Charles Morris. *Signs, Language, and Behavior*. Prentice-Hall, 1946.

[Mos01] Peter D Mosses. “The varieties of programming language semantics and their uses”. In: *International Andrei Ershov Memorial Conference on Perspectives of System Informatics*. Springer. 2001, pp. 165–190.

[Mos74] Peter David Mosses. *The mathematical semantics of ALGOL60*. Tech. rep. Programming Research Group, 1974. url: http : / / homepages . cs . ncl . ac . uk / cliff.jones/publications/OCRd/Mosses74.pdf.

[Mos75] Peter David Mosses. “Mathematical semantics and compiler generation”. PhD thesis. University of Oxford, 1975.

[MP67] John McCarthy and James A. Painter. “Correctness of a compiler for arithmetic expressions”. In: *Mathematical aspects of computer science* 19 (1967).

[MS74] Robert Milne and Christopher Strachey. *A Theory of Programming Language Semantics*. Privately circulated. Submitted for the Adams Prize. 1974.

[MS76] R. Milne and C. Strachey. *A Theory of Programming Language Semantics (Parts A and B)*. Chapman and Hall, 1976.

[Nau81] Peter Naur. “The European side of the last phase of the development of ALGOL 60”. In: *History of programming languages*. Ed. by Richard L. Wexelblat. Academic Press, 1981. Chap. 3, pp. 92–137.

[Neu16] Erich J. Neuhold. *Interview with Erich J. Neuhold*. Conducted by Troy Astarte and Cliff Jones. 2016.

[NPA14] David Nofre, Mark Priestley, and Gerard Alberts. "When Technology Became Language: The Origins of the Linguistic Conception of Computer Programming, 1950–1960". In: *Technology and Culture* 55.1 (2014), pp. 40–75.

[Pai67] J. A. Painter. *Semantic Correctness of a Compiler for an Algol-like Language*. Tech. rep. AI Memo 44. Computer Science Department, Stanford University, 1967.

[Pec71] J. E. L. Peck. *Minutes of the 13th meeting of IFIP TC2*. Held in the Ershov archive (online). Chaired by J. Peck. Held in Ljubljana. 1971. url: http://ershov.iis.nsk.su/en/node/806010.

[Per81] Alan J. Perlis. "The American Side of the Development of ALGOL". In: *History of programming languages*. Ed. by Richard L. Wexelblat. Academic Press, 1981. Chap. 3, pp. 75–91.

[Pet15] Elizabeth R Petrick. *Making computers accessible: Disability rights and digital technology*. JHU Press, 2015.

[Plo18] Gordon D. Plotkin. *Interview with Gordon D. Plotkin*. Conducted by Troy Astarte and Cliff Jones. 2018.

[Pnu79] A. Pnueli. "The Temporal Semantics of Concurrent Programs". In: *Proc. Eighteenth Annual Symposium on Semantics of Concurrent Computation*. Ed. by G. Kahn. Vol. 70. Lecture Notes in Computer Science. Heidelberg: Springer-Verlag, 1979, pp. 1–20.

[Pri11] Mark Priestley. *A Science of Operations: Machines, Logic and the Invention of Programming*. Springer Science & Business Media, 2011.

[Pri19] Giuseppe Primiero. *On the Foundations of Computing*. Oxford University Press, USA, 2019.

[PV88] María Eloína Peláez Valdez. "A gift from Pandora's box: The software crisis." PhD thesis. University of Edinburgh, 1988.

[Rus31] Bertrand Russell. *The Scientific Outlook*. London: George Allen and Unwiri, 1931.

[RW12] Bertrand Russell and Alfred North Whitehead. *Principia Mathematica*. Vol. 2. University Press, 1912.

[SB79] Claire Stegmann and John W. Backus. "Pathfinder". In: *Think* 45.4 (1979).

[Sch00] David A Schmidt. "Induction, domains, calculi: Strachey's contributions to programming-language engineering". In: *Higher-Order and Symbolic Computation* 13.1-2 (2000), pp. 89–101.

[Sco00] Dana S. Scott. "Some reflections on Strachey and his work". In: *Higher-Order and Symbolic Computation* 13.1 (2000), pp. 103–114.

[Sco16] Dana S. Scott. *Greetings to the Participants at "Strachey 100"*. A talk read out at the Strachey 100 centenary conference. Available online in the conference booklet. 2016. url: http://www.cs.ox.ac.uk/strachey100/Strachey_booklet.pdf.

[Sco77] Dana S Scott. "Logic and programming languages". In: *Communications of the ACM* 20.9 (1977), pp. 634–641.

[Spi88] J. M. Spivey. *Understanding Z—A Specification Language and its Formal Semantics*. Cambridge Tracts in Computer Science 3. Cambridge University Press, 1988.

[Ste66] T. B. Steel. *Formal Language Description Languages for Computer Programming*. North-Holland, 1966.

[Sto16a] Joseph E. Stoy. *Christopher Strachey and the Programming Research Group*. Talk given at the Strachey 100 symposium, Oxford University. 2016.

[Sto16b] Joseph E. Stoy. *Interview with Joseph E. Stoy*. Conducted by Troy Astarte. 2016.

[Str66] C. Strachey. "Towards a Formal Semantics". In: *[Ste66]*. North-Holland, 1966.

[Str71a] Christopher Strachey. *Curriculum Vitae*. Christopher Strachey Collection, Bodleian Library, Oxford. Box 248, A.3. Written by Strachey to send to the Times newspaper in case of the need for obitual information. 1971.

[Str71b] Christopher S. Strachey. *An Abstract Model for Stor- age (1st Draft)*. Tech. rep. Oxford University Programming Research Group, 1971.

[Tar44] Alfred Tarski. "The semantic conception of truth: and the foundations of semantics". In: *Philosophy and phenomenological research* 4.3 (1944), pp. 341–376.

[Ted14] Matti Tedre. *The science of computing: shaping a discipline*. Chapman and Hall/CRC, 2014.

[Tur19] David A Turner. *Some History of Functional Programming Languages*. Peter Landin Semantics Seminar. Dec. 2019. url: https://youtu.be/ezFZIPuSQU8.

[Tur67] W. M. Turski. *Minutes of the 8th meeting of IFIP WG 2.1*. Online. Held in Zandvoort, Netherlands. Chaired by W. L. van der Poel. Archived by Andrei Ershov. 1967. url: http://ershov.iis.nsk.su/en/node/778136.

[Utm63] R. E. Utman. *Minutes of the 3rd meeting of IFIP TC2*. Held in the Ershov archive (online). Chaired by H. Zemanek. Held in Oslo. 1963. url: http://ershov.iis.nsk.su/en/node/805662.

[Utm64a] R. E. Utman. *Minutes of the 3rd meeting of IFIP WG 2.1*. Online. Held in Tutzing, West Germany. Chaired by W. L. van der Poel. Archived by Andrei Ershov. 1964. url: http://ershov.iis.nsk.su/en/node/778087.

[Utm64b] R. E. Utman. *Minutes of the 4th meeting of IFIP TC2*. Held in the Ershov archive (online). Chaired by H. Zemanek. Held in Liblice. 1964. url: http://ershov.iis.nsk.su/en/node/805663.

[Wad00] Christopher P. Wadsworth. "Continuations revisited". In: *Higher-Order and Symbolic Computation* 13.1-2 (2000), pp. 131–133.

[Wal02] Kurt Walk. "Roots of Computing in Austria: Contributions of the IBM Vienna Laboratory and Changes of Paradigms and Priorities in Information Technology". In: *Human Choice and Computers*. Springer, 2002, pp. 77–87.

[Wal68] Kurt Walk. *Minutes of the 2nd meeting of IFIP WG 2.2 on Formal Language Description Languages*. Held in Vedbaek-Copenhagen, Denmark. Chaired by T. B. Steel. 1968.

[Wal69] Kurt Walk. *Minutes of the 3rd meeting of IFIP WG 2.2 on Formal Language Description Languages*. Held in Vienna, Austria. Chaired by T. B. Steel. 1969.

[Weg70] Peter Wegner. "Three Computer Cultures: Computer Technology, Computer Mathematics, and Computer Science". In: *Advances in Computers* 10 (1970), pp. 7–78. doi: 10.1016/S0065-2458(08)60431-3.

[Weg72] Peter Wegner. "The Vienna definition language". In: *ACM Computing Surveys (CSUR)* 4.1 (1972), pp. 5–63.

[Weg76] Peter Wegner. "Research paradigms in computer science". In: *Proceedings of the 2nd international conference on Software engineering*. IEEE Computer Society Press. 1976, pp. 322–330.

[Wex81] Richard L. Wexelblat, ed. *History of programming languages*. Academic Press, 1981.

[Wij62] Adriaan van Wijngaarden. "Generalized ALGOL". In: *Symbolic Languages in Data Processing, Proc. Symp. Intl, Computation Center Rome*. 1962, pp. 409–419.

[WMPK68] Adriaan van Wijngaarden, B. J. Mailloux, J. E. L. Peck, and C. H. A. Koster. *Draft report on the Algo- rithmic Language ALGOL 68*. Report MR 93. Mathe- matisch Centrum, 1968.

[WMPK69] A. van Wijngaarden, B. J. Mailloux, J. E. L. Peck, and C. H. A. Koster. *Report on the Algorithmic Language*

ALGOL 68. Second printing , MR 101. Mathematisch Centrum, Amsterdam, 1969.

[WWG51] Maurice Vincent Wilkes, David J. Wheeler, and Stan- ley Gill. *The preparation of programs for an electronic digital computer: with special reference to the EDSAC and the use of a library of subroutines*. Addison-Wesley Press, 1951.

[ZA87] Heinz Zemanek and William Aspray. *Interview: Heinz Zemanek*. Typescript. 1987.

[Zem65] H. Zemanek. *Semiotics and Programming Languages*. Tech. rep. 25.057. Also published in CACM 9.3 (Mar 1966). IBM Laboratory Vienna, 1965.

[Zem80] Heinz Zemanek. "Abstract Architecture". In: *Abstract Software Specifications: 1979 Copenhagen Winter School January 22 – February 2, 1979 Proceedings*. Ed. by Dines Bjørner. Vol. 86. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 1980. isbn: 978-3-540-38136-5.

[ULD66] ULD-III. *Formal Definition of PL/I (Universal Language Document No. 3)*. Tech. rep. 25.071. Author given as 'PL/I – Definition Group of the Vienna Laboratory'. IBM Laboratory Vienna, 1966.

[Łuk85] Leon Łukaszewicz. "A Handful of Recollections about IFIP People". In: *A Quarter Century of IFIP*. Ed. by H. Zemanek. North Holland, 1985, pp. 295–299.